Tutorial for VR-OpenMASK, a Software Development Platform for Virtual Reality

Thierry Duval

BUNRAKU Project – IRISA

Thierry.Duval@irisa.fr
Objectives

➢ Simulation, animation and interaction within 3D shared virtual universes...

✔ Collaborative Virtual Environments (CVE)
Application fields

- 3D Modeling and collaborative prototyping
- Collaborative virtual project review
- Distant demonstrations of complex manipulations upon virtual mockups/prototypes
- Multi-users virtual environments for learning:
  - A teacher with one or several students
  - Collaborative tasks
Main problems within VE and CVE

➢ How can we visualize virtual universes?
➢ How can we navigate within virtual universes?
➢ How can we interact with virtual objects?
➢ How can we share virtual universes?
➢ How can we share interactions?
Our solution: VR-OpenMASK

➢ A software development platform for collaborative virtual reality

✔ OpenMASK C++ object-oriented kernel

✗ Scheduling of a set of agents (multi-agents system)

✔ 3D Visualization (OpenSG or Ogre 3D)

✔ Virtual tools for interaction

✔ Networking with PVM

➢ Open source

✔ www.openmask.org
VR-OpenMASK : 4 points of view

➢ The point of view of a project manager

✔ Can VR-OpenMASK be a solution to his problems?

➢ The point of view of a VR-OpenMASK user

✔ How can he install VR-OpenMASK?
✔ How can he use existing VR-OpenMASK entities?

➢ The point of view of a VR-OpenMASK developer

✔ How can he develop new VR-OpenMASK entities?

➢ The point of view of a VR-OpenMASK contributor

✔ How can he develop new VR-OpenMASK utilities?
From a project manager point of view

➢ What can be done with OpenMASK ?
  ✔ Multi-frequencies simulations
  ✔ Distributed simulations

➢ What is VR-OpenMASK ?
  ✔ VR-OpenMASK : OpenMASK + VR features

➢ What can be done with VR-OpenMASK ?
  ✔ Interactive Virtual Environments
  ✔ Collaborative Virtual Environments
Simulation with OpenMASK

- A simulation is defined by a set of entities
- Each entity has its own frequency
- A Controller is in charge of the scheduling
- Entities communicate by:
  - Data flow connections
  - Events and messages sendings
- The kernel is in charge of these communications
Single process simulation

Node 1

T8:Tracker
T7:Tracker
T6:Tracker
T5:Tracker
T4:Tracker
T3:Tracker
T2:Tracker
T1:Trajectory
V1:Visualization
Multi-frequencies simulations
Distributed simulations

- A simulation can use:
  - Several processes
  - Several machines

- Each entity is associated to a process

- Distribution is useful:
  - To distribute computational weight
  - To use device drivers on dedicated machines
  - To share a simulation between several users...
Delegated visualization

Node 1

T8:Tracker
T7:Tracker
T6:Tracker
T5:Tracker
T4:Tracker
T3:Tracker
T2:Tracker
T1:Trajectory

Node 2

M T8:Tracker
M T7:Tracker
M T6:Tracker
M T5:Tracker
M T4:Tracker
M T3:Tracker
M T2:Tracker

V1:Visualization
Distribution of the computations

Node 1
- T1: Trajectory
- T2: Tracker
- T3: Tracker
- T4: Tracker

Node 2
- T5: Tracker
- T6: Tracker
- T7: Tracker
- T8: Tracker

Node 3
- M_T2: Tracker
- M_T3: Tracker
- M_T4: Tracker
- M_T5: Tracker
- M_T6: Tracker
- M_T7: Tracker
- M_T8: Tracker

V1: Visualization
VR-OpenMASK: OpenMASK + Virtual Reality features

- The 3D Visualizer: an entity dedicated to graphics
  - Visualizes all the graphical simulation objects
  - Can be instantiated several times to allow collaboration

- Data-types dedicated to animation

- Low-level drivers to encapsulate physical devices

- High-level interactors to interact with entities

- Adaptors to make entities interactive

- Tools to make 3D navigation easy
Collaboration

➢ Several interactors, located on different processes, handled by different end-users, can be used simultaneously within the same virtual world

➢ To control different objects
  ✔ One user per object

➢ To control the same object
  ✔ Several users per object
  ✔ This object integrates all the “inputs”
Typical collaborative simulation

Node 1:
- T1: Trajectory
- T2: Tracker
- T3: Tracker
- T4: Tracker

Node 2:
- T5: Tracker
- T6: Tracker
- T7: Tracker
- T8: Tracker

Node 3:
- M_T1: Tracker
- M_T2: Tracker
- M_T3: Tracker
- M_T4: Tracker

Node 4:
- V1: Visualization
- V2: Visualization

- M_T5: Tracker
- M_T6: Tracker
- M_T7: Tracker
- M_T8: Tracker
Sharing a dynamical CVE
Sharing a virtual mockup
From a project manager point of view

- VR-OpenMASK enables to create virtual worlds:
  - Filled with autonomous entities
  - Distributed
  - Interactive
  - Collaborative
  - Dynamical

- VR-OpenMASK:
  - A good tool for Collaborative Virtual Environments
From a user point of view

➢ How can we install VR-OpenMASK ?
➢ How can we use existing applications ?
➢ How can we customize applications ?
➢ How can we basically use the 3D Visualizer ?
➢ How can we navigate within virtuals worlds ?
➢ How can we share virtual worlds ?
Installing VR-OpenMASK (1/2)

➢ Download OpenMASK at www.openmask.org

➢ Download also:

✔ PCCTS : to generate the loaders
✔ PVM : for networked simulations
✔ OpenSG : for the 3D Visualization
✔ gcc 3.3.6 or gcc 4
✔ GLUT 3.7 : if not installed on the system...
✔ libcwiid 0.0.6 : the wiimote driver...
Installing VR-OpenMASK (2/2)

➢ Set the environment variables:

✔ OpenMASKDIR
✔ PVM_ROOT
✗ PVM_ARCH, PVM_DPATH, PVM_EXPORT, PVM_RSH
✔ OSGROOT
✔ PCCTSDIR, PCCTSBINDIR
✔ LD_LIBRARY_PATH, (PATH)
✔ (COMPILER)

➢ Compile the kernel and your application...

➢ Have a look at the documentation:

✔ /usr/local/EVC/OpenMASK/Documentation/html
IFSIC OpenMASK environment: .bashrc

export PVM_ROOT=/usr/local/EVC/pvm3
export PVM_ARCH=`$PVM_ROOT/lib/pvmgetarch`
export PVM_DPATH=$PVM_ROOT/lib/pvmd

export PATH=$PVM_ROOT/bin/$PVM_ARCH:$PVM_ROOT/lib:$PATH
export PATH=$PVM_ROOT/lib/$PVM_ARCH:$PATH
export PVM_EXPORT=DISPLAY:PATH

export PCCTSDIR=/usr/local/EVC/pccts
export PCCTSBINDIR=$PCCTSDIR/bin

export OSGROOT=/usr/local/EVC/OpenSG
export OpenMASKDIR=/usr/local/EVC/OpenMASK

export LD_LIBRARY_PATH=$OpenMASKDIR/lib/opt:$OSGROOT/lib/opt:../$PVM_ARCH
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/usr/local/EVC/cwiid-0.6.00/lib

export PVM_RSH=/usr/bin/ssh
Looking at an application “main” file

- Construction of the simulation tree:
  - Directly in the main file
  - Or thanks to a simulation configuration file
- Creation of the controller
- Declaration of the class symbolic names of the simulated objects:
  - To teach the controller how to create simulated objects
- Initialization of the controller with the simulation tree
- Running of the controller
// the simulation tree loader
#include <SimpleOpenMASK3Loader.h>
#include <PsController.h>
#include <PsException.h>

// PVM
#include <PsPvmController.h>
#include <PsnPvmSvm.h>
#include <PsnPvmProcess.h>
#include <pvm3.h>

// 3D Visualization
#include <Sensitive3DVis.h>

// addings to obtain trajectory and trackers
#include <TrajectoryForTutorial.h>
#include <TrackerForTutorial.h>
int main (int argc, char * argv []) {
    if(argc >= 2) {
        PsObjectDescriptor * simulationTree;
        SimulationTree =
            (new SimpleOpenMASK3Loader (argv[1]))
                ->getRootObjectDescriptor () ;
        PsController * controller ;
        char* extension = strrchr (argv[1], '.') ;
        if (extension != NULL && strcmp (extension, "\0") == 0) {
            std::cout << "Mode Multi-processus" << std::endl ;
            controller = new PsPvmController (*simulationTree, 0, argc, argv) ;
        } else {
            std::cout << "Mode Mono-processus" << std::endl ;
            controller = new PsController (*simulationTree, 0) ;
        }
    }
// VR-OpenMASK visualization
controller->addInstanceCreator("Sensitive3DVis",
    new PsSimpleSimulatedObjectCreator<Sensitive3DVis>());

// minimal state, for trajectory, and trackers
controller->addInstanceCreator("TrajectoryForTutorial",
    new PsSimpleSimulatedObjectCreator<TrajectoryForTutorial>());
controller->addInstanceCreator("TrackerForTutorial",
    new PsSimpleSimulatedObjectCreator<TrackerForTutorial>());

// init of the controller and run
try {
    controller->init();
    controller->run();
}
catch (PsException & e) {
    std::cerr << "Unresolved exception " << e << std::endl;
}
Compiling the application

➢ Thanks to its associated “Makefile”

✔ make -f MinimalMakefile
✔ Result : the “LINUX/minimalTutorial” binary file
include $(OpenMASKDIR)/make.in

COMPILE = $(COMPILER) $(COMPILEFLAGS) -g -fpermissive
LINK = $(LINKER) -WI,-noinhibit-exec -L/local/cwiid-0.6.00/libcwiid/lib -lcwiid

INCDIR = -I. \ 
         -I/usr/local/EVC/cwiid-0.6.00/libcwiid/include \ 
         $(shell $(OpenMASKDIR)/bin/omk-config --cxxflags)

ifdef PVM_ROOT
  PVMLIBS = -L$(PVM_ROOT)/lib/$(PVM_ARCH) \ 
            -lOpenMASKPvm -lgpvm3 -lpvm3
  INCDIR += -I$(PVM_ROOT)/include
endif

DSO = $(shell $(OpenMASKDIR)/bin/omk-config --ldflags) \ 
      -L$(PVM_ROOT)/lib/$(PVM_ARCH)
TARGETNAME = ./$(PVM_ARCH)/minimalTutorial

$(TARGETNAME): $(OBJ) $(PVM_ARCH)/minimalMain.o
  $(LINK) $(OBJ) $(PVM_ARCH)/minimalMain.o $(DSO) -o $@

$(PVM_ARCH)/minimalMain.o : minimalMain.cxx
  $(COMPILE) $(INCDIR) -g -c minimalMain.cxx -o $@

all:
  make $(TARGETNAME)

clean:
  rm -rf ./$(PVM_ARCH)/*.o

cleanall:
  rm -rf ./$(PVM_ARCH)/*.o $(TARGETNAME)
Running the application

➢ Using a configuration file:
  ✓ minimalTutorial tutorial.minimalTrajectory.monop
#OpenMASK3

root {
    Class Controller
    Sons {
        include "MinimalVisCVI.processA"
        include "Trajectory.processA"
        include "MinimalTrackers.processA"
    }
}
visA {
    Class Sensitive3DVis
    Scheduling {
        Frequency 75
        Process visualProcessA
    }
}
UserParams {
    interactivePipeNumber 0
    pipes {
        pipe0 {
            screenNumber 0
            windows {
                window0 {
                    windowName "Green Cabin window"
                    fullScreen false
                    origin [ 0 0 ]
                    size [ 800 600 ]
                    viewports {
                        viewport0 {
                            displayStatistics false
                        }
                    }
                }
            }
        }
    }
}
trajectory {
    Class TrajectoryForTutorial
    Scheduling {
        Frequency 5
        Process visualProcessA
    }
    UserParams {
        step 0.25
        trajectory [ [-60 -40 -150] [-60 40 -150] [-50 40 -100] [-50 -40 -100] 
                    [50 0 -100] [60 40 -150] [0 0 -100] [0 0 -50] [0 0 0] [0 0 50] ]
    }
}
MinimalTrackers.processA (1/2)

tracker1 {
  Class TrackerForTutorial
  Scheduling {
    Frequency 75
    Process visualProcessA
  }
  UserParams {
    geometryFileName ../../../Data/RedCone.wrl
    direction [0 1 0] // RedCone's front is Y+
    target [trajectory position]
    slowingFactor 100
  }
}

tracker2...
...
tracker10...
tracker11 {
    Class TrackerForTutorial
    Scheduling {
        Frequency 75
        Process visualProcessA
    }
    UserParams {
        geometryFileName ../../../Data/TieFighter.wrl
direction [0 0 1] // TieFighter's front is Z+
target [tracker11 position]
slowingFactor 100
    }
}
}

tracker12...
...
tracker20...
Sharing virtual worlds

➢ For each machine of the simulation, explain in a PVM hosts file where things can be found:
  ✔ wd=WorkingDirectory
  ✔ ep=ExecutionPath
  ✔ dx=PlaceOfThePVMDaemon
  ✔ lo=UserLogin

➢ Run, on the same machine:
  ✔ the PVM machine with the hosts file
  ✔ the application
Running the application using two process

➢ Using PVM :
  ✔ pvm hosts

➢ Using a configuration file :
  ✔ minimalTutorial tutorial.minimalTrajectory.multi 0
The PVM file : hosts

➤ Warning : everything on the same line !
✔ One line per machine...

* wd=/usr/local/EVC/Tutorial2008/Execution
  ep=/usr/local/EVC/Tutorial2008/Execution:$PVM_ROOT/bin/LINUX
  dx=$PVM_ROOT/lib/pvmd

vapor3d wd=/local/tduval/EVC/Tutorial2008/Execution
  ep=/local/tduval/EVC/Tutorial2008/Execution:$PVM_ROOT/bin/LINUX
  dx=$PVM_ROOT/lib/pvmd
root {
    Class Controller
    Scheduling {
        Latency 20
        Machines {
            visualProcessA vapor3d
            visualProcessB vapor3d
        }
    }
    Sons {
        include "MinimalVisCVI.processA"
        include "MinimalVisCVI.processB"
        include "Trajectory.processA"
        include "MinimalTrackers.processA"
    }
}
Customizing/building applications

- By creating new configuration files
- By building new applications:
  - New “main” files with associated “Makefile” files
  - Use of existing simulated objects libraries
Using the 3D visualizer and the IVC

- Defining the characteristics of the display:
  - The number of graphic pipes
  - The number of windows
  - The viewports in the windows:
    - Their graphical characteristics (clipping, fov, size, ...)
    - Their associated viewpoint (a cameraman supported by the IVC)
    - The camera of the cameraman (it can own several cameras)

- Thanks to:
  - A new main file
  - New configuration files...
/ the simulation tree loader
#include <SimpleOpenMASK3Loader.h>
#include <PsController.h>
#include <PsException.h>

// PVM
#include <PsPvmController.h>
#include <PsnPvmSvm.h>
#include <PsnProcess.h>
#include <pvm3.h>

// 3D Visualization
#include <Sensitive3DVis.h>
// addings for interactions
#include <PsXEventHandlerFilterForTutorial.h>

// addings to obtain trajectory and trackers
#include <TrajectoryForTutorial.h>
#include <TrackerForTutorial.h>

// for the Immersive Virtual Cabin
#include <IVC.h>
#include <ScalableSupportedOnPO.h>
int main (int argc, char * argv []) {
    if(argc >= 2){
        PsObjectDescriptor * simulationTree ;
        SimulationTree =
            (new SimpleOpenMASK3Loader (argv[1]))
        ->getRootObjectDescriptor () ;
        PsController * controller ;
        char* extension = strrchr (argv [1], '.') ;
        if (extension != NULL && strcmp (extension, ".multi") == 0) {
            std::cout << "Mode Multi-processus" << std::endl ;
            controller = new PsPvmController (*simulationTree, 0, argc, argv) ;
        } else {
            std::cout << "Mode Mono-processus" << std::endl ;
            controller = new PsController (*simulationTree, 0) ;
        }
    }
// VR-OpenMASK utility to filter graphic events
controller->addInstanceCreator("GraphicEventHandlerFilter",
    new PsSimpleSimulatedObjectCreator<
        PsXEventHandlerFilterForTutorial> ()) ;

// VR-OpenMASK visualization
controller->addInstanceCreator("Sensitive3DVis",
    new PsSimpleSimulatedObjectCreator<Sensitive3DVis> ()) ;

// basic CVI objects
controller->addInstanceCreator("ImmersiveVirtualCabin",
    new PsSimpleSimulatedObjectCreator<IVC> ()) ;
controller->addInstanceCreator("ScalableCameraman",
    new PsSimpleSimulatedObjectCreator<ScalableSupportedOnPO> ()) ;
// minimal state, for trajectory, and trackers
controller->addInstanceCreator("TrajectoryForTutorial",
    new PsSimpleSimulatedObjectCreator<TrajectoryForTutorial>());
controller->addInstanceCreator("TrackerForTutorial",
    new PsSimpleSimulatedObjectCreator<TrackerForTutorial>());

// init of the controller and run
try {
    controller->init();
    controller->run();
} catch (PsException & e) {
    std::cerr << "Unresolved exception " << e << std::endl;
}
}
Using the IVC

- Defining its main characteristics:
  - Its position and orientation: x y z h p r
  - Its associated graphic visualization: a VRML file
  - Its translation step increment
  - Its rotation step increment
Using the cameraman

- Defining the characteristics of the viewpoint:
  - Its position and orientation: x y z h p r
  - Relative to its support: probably an IVC...
  - Its associated graphic visualization: a VRML file
  - Its associated graphic visualization: a VRML file
  - Its cameras: within its associated VRML file
# OpenMASK3

root {
    Class Controller
    Sons {
        include "SimpleVisCVI.processA"
        include "Trajectory.processA"
        include "MinimalTrackers.processA"
    }
}
visA {
  Class Sensitive3DVis
  Scheduling {
    Frequency 75
    Process visualProcessA
  }
}
UserParams {
  interactivePipeNumber 0
  pipes {
    pipe0 {
      screenNumber 0
      windows {
        window0 {
          windowName "Green Cabin window"
          fullScreen false
          origin [ 0 0 ]
          size [ 800 600 ]
          viewports {
            viewport0 {
              displayStatistics false
              associatedObservable "greenCameraman"
            } }
        } }
      viewports {
      }
    } }
}
SimpleVisA.processA (3/4)

greenCabin {
    Class ImmersiveVirtualCabin
    Scheduling {
        Frequency 75
        Process visualProcessA
    }
    UserParams {
        geometryFileName ../../../Data/GreenIVC.wrl
        positionOrientationXYZHPR [ 0 0 100 0 0 0 ]
        navigationStep 1
        rotationStep 5
    }
}
SimpleVisA.processA (4/4)

greenCameraman {
    Class ScalableCameraman
    Scheduling {
        Frequency 75
        Process visualProcessA
    }
    UserParams {
        geometryFileName ../../Data/GreenCameraman.wrl
        clone false
        positionOffset [ 0 1.75 0 ]
        orientationOffset [ 0 0 0 ]
        positionOrientationToFollow [ greenCabin positionOrientation ]
        scaleToFollow [ greenCabin scale ]
    }
}
Somewhere within the VRML File

... 
DEF DCS_positionOrientation Transform {
  children [
    DEF DCS_CAMERA1 Transform {
      translation 0 10 -4.4
    }
    DEF DCS_CAMERA2 Transform {
      translation 0 8 -4.4
    }
    DEF DCS_CAMERA3 Transform {
      translation 0 0 0
      rotation 0 1 0 10
    }
  ]
...
3D Visualizer with 4 Viewports
multipleVisA {
Class Interactive3DVis
Scheduling {
  Frequency 75
  Process visualProcessA
}
UserParams {
  pipes {
    pipe0 {
      screenNumber 0
      windows {
        window0 {
          windowName "window A"
          fullScreen false
          origin [ 0 0 ]
          size [ 800 600 ]
        }
      }
    }
  }
}
viewports {
    viewport0 {
        associatedObservable "greenCameraman"
        viewportSize [0.0 0.5 0.0 0.5]
    }
    viewport1 {
        associatedObservable "yellowCameraman"
        viewportSize [0.5 1.0 0.0 0.5]
    }
    viewport2 {
        associatedObservable "redCameraman"
        viewportSize [0.0 0.5 0.5 1.0]
    }
    viewport3 {
        associatedObservable "blueCameraman"
        viewportSize [0.5 1.0 0.5 1.0]
    }
}
greenCabin {
    Class ImmersiveVirtualCabin
    Scheduling {
        Frequency 75
        Process visualProcessB
    }
    UserParams {
        geometryFileName ../../Data/GreenIVC.wrl
        positionOrientationXYZHPR [-10 0 -170 0 0 -180]
        tracePositionAndOrientation no
        navigationStep 1
        rotationStep 5
    }
}
MultipleVisCVI.processA (4/10)

greenCameraman {
    Class ScalableCameraman
    Scheduling {
        Frequency 75
        Process visualProcessA
    }
    UserParams {
        geometryFileName ../../Data/GreenCameraman.wrl
        clone false
        positionOffset [ 0 1.75 0 ]
        orientationOffset [ 0 0 0 ]
        navigationStep 1
        positionOrientationToFollow [ greenCabin positionOrientation ]
        scaleToFollow [ greenCabin scale ]
    }
}
yellowCabin {
    Class ImmersiveVirtualCabin
    Scheduling {
        Frequency 75
        Process visualProcessA
    }
    UserParams {
        geometryFileName ../../../Data/YellowIVC.wrl
        positionOrientationXYZHPR [0 0 10 0 0 0]
        tracePositionAndOrientation no
        navigationStep 1
        rotationStep 5
    }
}
yellowCameraman {
    Class ScalableCameraman
    Scheduling {
        Frequency 75
        Process visualProcessA
    }
    UserParams {
        geometryFileName ../../Data/YellowCameraman.wrl
        clone false
        positionOffset [ 0 1.75 0 ]
        orientationOffset [ 0 0 0 ]
        navigationStep 1
        positionOrientationToFollow [ yellowCabin positionOrientation ]
        scaleToFollow [ yellowCabin scale ]
    }
}
redCabin {
    Class ImmersiveVirtualCabin
    Scheduling {
        Frequency 75
        Process visualProcessA
    }
    UserParams {
        geometryFileName ../../Data/RedIVC.wrl
        positionOrientationXYZHPR [50 0 0 0 0 90]
        tracePositionAndOrientation no
        navigationStep 1
        rotationStep 5
    }
}
redCameraman {
    Class ScalableCameraman
    Scheduling {
        Frequency 75
        Process visualProcessA
    }
    UserParams {
        geometryFileName ../../../Data/RedCameraman.wrl
        clone false
        positionOffset [ 0 1.75 0 ]
        orientationOffset [ 0 0 0 ]
        navigationStep 1
        positionOrientationToFollow [ redCabin positionOrientation ]
        scaleToFollow [ redCabin scale ]
    }
}
blueCabin {
  Class ImmersiveVirtualCabin
  Scheduling {
    Frequency 75
    Process visualProcessA
  }
  UserParams {
    geometryFileName ../../Data/RedIVC.wrl
    positionOrientationXYZHPR [-10 0 40 0 0 0]
    tracePositionAndOrientation no
    navigationStep 1
    rotationStep 5
  }
}
blueCameraman {
    Class ScalableCameraman
    Scheduling {
        Frequency 75
        Process visualProcessA
    }
    UserParams {
        geometryFileName ../../../Data/BlueCameraman.wrl
        clone false
        positionOffset [ 0 1.75 0 ]
        orientationOffset [ 0 0 0 ]
        navigationStep 1
        positionOrientationToFollow [ blueCabin positionOrientation ]
        scaleToFollow [ blueCabin scale ]
    }
}
3D visualizer with 2 viewpoints
DoubleVisCVI.processA (1/3)

DoubleVisA {
    Class Sensitive3DVis
    Scheduling {
        Frequency 75
        Process visualProcessA
    }
    UserParams {
        pipes {
            pipe0 {
                screenNumber 0
                windows {

            
        
    
}
window0 {
    windowName "green window"
    fullScreen false
    origin [ 0 0 ]
    size [ 800 600 ]
    viewports {
        viewport0 {
            associatedObservable "greenCameraman"
        }
    }
}
window1 {
    windowName "yellow window"
    fullScreen false
    origin [ 800 600 ]
    size [ 800 600 ]
    viewports {
        viewport0 {
            associatedObservable "yellowCameraman"
        }
    }
}

...
Navigating thanks to the visualizer

- Navigation is interaction with:
  - The interactive 3D visualizer
  - An Immersive Virtual Cabin
  - Thanks to a device:
    - For example a keyboard input handler
  - Default actions such as switching the camera from one support to another, or moving the IVC:
    - Are associated to keys
    - Can be overridden by the end-user
tutorial.simpleTrajectory.monop

#OpenMASK3

root {
    Class Controller
    Sons {
        include "SimpleVisCVI.processA"
        include "Trajectory.processA"
        include "MinimalTrackers.processA"
    }
}
visA {
    Class Sensitive3DVis
    Scheduling {
        Frequency 75
        Process visualProcessA
    }
    UserParams {
        interactivePipeNumber 0
        pipes {
            pipe0 {
                screenNumber 0
                windows {
                
```
window0 {
    windowName "Green Cabin window"
    fullScreen false
    origin [ 0 0 ]
    size [ 800 600 ]
    viewports {
        viewports {
            viewport0 {
                displayStatistics false
                associatedObservable "greenCameraman"
            }
        }
    }
}
xEventHandlerA {
    Class GraphicEventHandlerFilter
    Scheduling {
        Frequency 75
        Process visualProcessA
    }
    UserParams {
        VisualisationName visA
        PressEvents [ 

{ Key "KEY_ESCAPE" Signal "Quit" To greenCabin }
{ Key "KEY_PAGE_UP" Signal "moveCameraman up" To greenCabin }
{ Key "KEY_PAGE_DOWN" Signal "moveCameraman down" To greenCabin }
{ Key "KEY_LEFT" Signal "moveCameraman left heading" To greenCabin }
{ Key "KEY_RIGHT" Signal "moveCameraman right heading"
  To greenCabin }
{ Key "KEY_DOWN" Signal "moveCameraman backward" To greenCabin }
{ Key "KEY_UP" Signal "moveCameraman forward" To greenCabin }
{ Key "KEY_HOME" Signal "moveCameraman up pitch" To greenCabin }
{ Key "KEY_END" Signal "moveCameraman down pitch" To greenCabin }
{ Key ":" Signal "moveCameraman left" To greenCabin }
{ Key "!" Signal "moveCameraman right" To greenCabin }
{ Key "KEY_INSERT" Signal "moveCameraman left roll" To greenCabin }
{ Key "KEY_DELETE" Signal "moveCameraman right roll" To greenCabin }
SimpleVisCVI.processA (5/5)

{ Key "b" Signal "becomeBigger" To greenCabin }
{ Key "B" Signal "becomeBigger" To greenWIM }
{ Key "s" Signal "becomeSmaller" To greenCabin }
{ Key "S" Signal "becomeSmaller" To greenWIM }
{ Key "g" Signal "becomeWellSized" To greenCabin }
{ Key "r" Signal "resetHP" To greenCabin }
{ Key "c" Signal "createWIMTarget" To greenCabin }
]
}
}
...

...
Example of Distributed Configuration File

root {
    Class Controller
    Scheduling {
        Latency 10
        Machines {
            visualProcessA malux.irisa.fr
            visualProcessB madredeux.irisa.fr
            visualProcessC rvmax-gb.irisa.fr
        }
    }
    Sons {
        ...
    }
}
Some useful OpenMASK classes

- PsXEventHandlerFilterForTutorial
  - Sends signals to named objects
  - Fires signals to registered objects

- XmouseTrigger
  - Sends signals to objects “under” the cursor of the mouse

- LocalClock
  - Is a local object:
    - There is no use to put it on each process (?)
  - Tries to obtain the right frequency for the global simulation
    - Able only to slow down the simulation
    - Its frequency should be the LCM of all the used frequencies...
Configuration example:

```plaintext
xEventHandlerMOA {
    Class GraphicEventHandlerFilter // the name declared in the main.cxx
    Scheduling {
        Frequency 75
        Process visualProcessA
    }
    UserParams {
        VisualisationName visA
        PressEvents [ 
            { Key "t" Signal "up" To MovableObject1 } 
            { Key "G" Signal "sendGo" } 
        ]
    }
}
```
XMouseTrigger

- **Configuration example:**

```java
xMouseTriggerTOA {
    Class XMouseTrigger
    Scheduling {
        Frequency 75
        Process visualProcessA
    }
    UserParams {
        associatedPicker visA
        Events [
            { Mousekey "middle" SignalToSend "askForTarget" }
            { Mousekey "right" SignalToSend "toggle" }
        ]
    }
}
```
LocalClock

- Configuration example:

LocalClockA {
    Class LocalClock
    Scheduling {
        Frequency 200
        Process visualProcessA
    }
    UserParams {
        ShowClock TRUE
        ClockPosition [0 400]
        ShowClockKey c
    }
}
LocalClock

Other configuration parameters:

/ * * ShowClock : TRUE | FALSE (TRUE)
  * ClockPosition : [int int] ([0 0])
  * ShowClockKey : char (c)
  * FasterKey : char (+)
  * SlowerKey : char (-)
  * FreeRunKey : char (*)
  * SimulClockColor : [float float float] ([.8 0 .8])
  * RealClockColor : [float float float] ([.9 .45 0])
  * MultCoeffColor : [float float float] ([0 .5 0])
  * DivCoeffColor : [float float float] ([.8 0 0])
  * InitCoeff : float (1.0)
  * FreeRun : TRUE | FALSE (FALSE)
  * SleepProcedure : "active" | "passive" ("active")
 */
From a user point of view

➢ He must install VR-OpenMASK...
➢ Then he can:

✔ Run existing applications
✔ Customize applications
✔ Build new applications by reusing modules
✔ Navigate within virtual worlds
✔ Share virtual worlds
From a programmer point of view

- What is OpenMASK?
- What is the API of a simulated object?
- How does the 3D Visualization work?
- How does the distribution works?
- How is it possible to develop new VR-OpenMASK entities?
OpenMASK – global view

- **Object-oriented approach (C++)**
- **Description of entities:**
  - ✔ Their API: inputs, outputs, control parameters
  - ✔ Their behavior: several methods to
    - ✗ initialize and finish the entity
    - ✗ process events and compute the new state of the entity
  - ✔ Their activation frequency (used by a scheduler)
- **A virtual world is described by its initial entities:**
  - ✔ Usually thanks to a configuration file...
The simulated object
PsTypes and interpolation

➢ Some provided PsTypes for attributes:
  ✔ PsTranslation, PsHPRRotation, PsScale, ...
  ✔ PsColor, PsHighlight
  ✔ PsXEvent, Ps3DVisEvent

➢ All PsTypes can be interpolated or extrapolated:
  ✔ Because simulated objects can have different frequencies

➢ Interpolation and extrapolation:
  ✔ Are provided by the kernel
  ✔ Can be customized for any PsType
The 3D visualization

- A viewer entity able to load 3D geometries:
  - Based on OpenSG, or on Ogre 3D
  - Can load VRML 2 or other formats (collada, mesh, ...)
- Some partners able to make animation links between outputs and graphics nodes:
  - PsvMechanismPartner
- Thanks to some input handlers:
  - 3D positions, 3D orientations, 3D scale, color, transparency...
The 3D visualization

3D Visualization

SO 1
- Position
- Orientation
- Behavior

SO 2
- Position
- Orientation
- Behavior

SO N
- Position
- Orientation
- Behavior

SO 1 Position
SO 1 Orientation

SO 2 Position
SO 2 Orientation

SO N Position
SO N Orientation

Behavior
Creating new simulated objects

- Inherit from the PsSimulatedObject class
  - Provide a new constructor

- Override some of the inherited methods:
  - init
  - compute
  - processEvent
    - Or use callbacks...
  - finish
Creating new visualizable objects

➢ Inherit also from a visualization partner:
  ✔ For example: PsvMechanismPartner
➢ Declare:
  ✔ Which outputs have to be visualized
  ✔ The name of their associated graphic node

visualiseOutput <PsvTranslationInputHandler> (positionOutput, "DCS_position") ;
visualiseOutput <PsvHPRRotationInputHandler> (orientationOutput, "DCS_orientation") ;
Typical constructor

- Creation of the attributes
- Initialization of references to the attributes
- Initialization of the outputs

```cpp
MyObject::MyObject (PsController & ctrl,
    const PsObjectDescriptor & objectDescriptor)
    : PsSimulatedObject (ctrl, objectDescriptor),
      PsvMechanismPartner (),
      targetInput (addInput<PsTranslation> ("target")),
      positionParameter (addControlParameter<PsTranslation> ("position")),
      positionOutput (addOutput<PsTranslation> ("position")) {
    visualiseOutput<PsvTranslationInputHandler> (positionOutput, "DCS_position") ;
}
```
Typical init

- Linking of the inputs to their associated outputs

```cpp
void MyObject::init (void) {
    // syntax : target [ targetName targetOutputName ]
    const PsConfigurationParameterDescriptor * configParamDescriptor =
        getConfigurationParameters ()->getSubDescriptorByName ("target") ;
    targetInput.connect (PsName (configParamDescriptor->
        getSubDescriptorByPosition (0)->getAssociatedString ()),
        PsName (configParamDescriptor->
        getSubDescriptorByPosition (1)->getAssociatedString ()));
    registerForSignal (PsEventIdentifier ("Suspend")) ;
}
```
Typical compute

- Reading of the inputs
- Computation of new parameter values
- Computation of new output values

```cpp
void MyObject::compute (void) {
    positionParameter.set (targetInput.get ()) ;
    ...
    positionOutput.set (positionParameter.get ()) ;
}
```
Typical `processEvent`

- Called once for each event to process...

```cpp
void MyObject::processEvent (PsEvent * event) {
    bool processed = false;
    if (event->eventId == "Suspend") {
        iAmActive = false;
        processed = true;
    } else if (event->eventId == "Resume") {
        iAmActive = true;
        processed = true;
    }
    return processed;
}
```

- The initial one manages callbacks...
Can be needed to be notified...

➢ Register to any signal:

```cpp
registerForSignal (PsEventIdentifier ("Suspend"));
```

➢ Register to a signal from an object:

```cpp
registerForSignalBy (PsEventIdentifier ("Suspend"), "SignalSource");
```

➢ Create a listener callback:

```cpp
new PsEventListenerCallBack<MyObjectType>(
    *this, &MyObjectType::theCallback, "theTrigger");
```

➢ Declare the callback:

```cpp
bool MyObjectType::theCallback (PsEvent * event) {
    bool processed = false;
    if (event->eventId == "theTrigger") {
        ...; processed = true;
    }
    return processed;
}
```
Can be used to notify...

- **Broadcast a signal:**
  - to objects which have registered to that signal...
    ```java
    fireSignal("Suspend");
    ```

- **Send an event to an object:**
  - the object does not need to register...
    ```java
    sendEvent("SignalTarget", "Suspend");
    ```
Typical finish

➢ Cleaning the context...

```cpp
void MyObject::finish (void) {
    cancelRegistrationForSignal (PsEventIdentifier ("Suspend")) ;
}
```
Accessing configuration parameters

➢ The file to include:

```cpp
#include <PsConfigurationParameterDescriptor.h>
```

➢ And how to use it:

```cpp
if (getConfigurationParameters ()
    ->getSubDescriptorByName ("IAmTargetOf") != NULL) {
    std::string IAmTargetOf = getConfigurationParameters ()
        ->getSubDescriptorByName ("IAmTargetOf")->getAssociatedString () ;
}
if (getConfigurationParameters ()
    ->getSubDescriptorByName ("active") != NULL) {
    std::string sActive = getConfigurationParameters ()
        ->getSubDescriptorByName ("active")->getAssociatedString () ;
    bool active = (sActive == "true") ;
}
```
Using dataflow

➢ What we need is:

✔ Objects with outputs
✔ Objects with inputs
✔ Links between inputs and outputs

➢ The kernel is in charge of transferring outputs values towards connected inputs

✔ Even with different frequencies!
✔ Even across the network!
Some simple OpenMASK objects

➢ The “TrajectoryForTutorial”
   ✓ One simple output:
     ✗ the current position within the trajectory

➢ The “TrackerForTutorial”
   ✓ Two visualizable outputs:
     ✗ it's position
     ✗ it's orientation
   ✓ One input:
     ✗ the position of the target of the tracker
#ifndef TrajectoryForTutorialHEADER
#define TrajectoryForTutorialHEADER

#include <PsSimulatedObject.h>
#include <PsTranslation.h>
#include <PsOutput.h>
#include <vector>
class TrajectoryForTutorial : public PsSimulatedObject {

public:

    TrajectoryForTutorial (PsController & ctrl,
                            const PsObjectDescriptor & objectDescriptor) ;
    virtual ~TrajectoryForTutorial (void) ;
    virtual void init (void) ;
    virtual void compute (void) ;
protected:

virtual void computeProgression (void);
virtual void computeCurrentPosition (const PsTranslation & previous,
                                         const PsTranslation & current,
                                         const float progression);

virtual void computeOutputs (void);
virtual PsTranslation getCurrentTarget (void);
virtual PsTranslation getPreviousTarget (void);
virtual float A2B (const float & a, const float & b, const float & f) const;
PsOutput<PsTranslation> & positionOutput;
int _numberOfTargets;
std::vector<PsTranslation> _targetsCoordinates;
float _step;
float _positionBetweenTargets;
int _currentTarget;
int _previousTarget;
PsTranslation _position;
} ;

#endif
#include <TrajectoryForTutorial.h>
#include <PsConfigurationParameterDescriptor.h>
#include <sstream>

TrajectoryForTutorial::TrajectoryForTutorial (PsController & ctrl,
    const PsObjectDescriptor & objectDescriptor)
    : PsSimulatedObject (ctrl, objectDescriptor),
    positionOutput (addOutput<PsTranslation> ("position")),
    _numberOfTargets (0),
    _step (0.05),
    _currentTarget (0),
    _positionBetweenTargets (1),
    _position (0, 0, 0) {

if (getConfigurationParameters () != NULL) {
    const PsConfigurationParameterDescriptor * configurationParameters ;
    configurationParameters = getConfigurationParameters ()
        ->getSubDescriptorByName ("step") ;
    if (configurationParameters != NULL) {
        istringstream is (configurationParameters->getAssociatedString ().c_str()) ;
        is >> _step ;
    }
    configurationParameters = getConfigurationParameters()
        ->getSubDescriptorByName ("trajectory") ;
    if (configurationParameters != NULL) {
        const PsConfigurationParameterDescriptor * targetParameters ;
        for (int i = 0 ; i < configurationParameters->getNumberOfSubItems () ; ++ i) {
targetParameters = configurationParameters->getSubDescriptorByPosition (i) ;
if (targetParameters->getNumberOfSubItems () == 3) {
    PsFloat x, y, z ;
    X = atof (targetParameters->getSubDescriptorByPosition (0)
                ->getAssociatedString ().c_str ()) ;
    y = atof (targetParameters->getSubDescriptorByPosition (1)
                ->getAssociatedString ().c_str ()) ;
    Z = atof (targetParameters->getSubDescriptorByPosition (2)
                ->getAssociatedString ().c_str ()) ;
    PsTranslation targetCoordinate (x, y, z) ;
    _targetsCoordinates.push_back (targetCoordinate) ;
    ++_numberOfTargets ;
} } } }
if (_numberOfTargets == 0) {
    _numberOfTargets = 2;
    _targetsCoordinates.push_back (PsTranslation (-50, 100, 1));
    _targetsCoordinates.push_back (PsTranslation (50, 100, 10));
}

_position = _targetsCoordinates [0];
positionOutput.set (_position);
}
TrajectoryForTutorial::~TrajectoryForTutorial (void) {
}

void TrajectoryForTutorial::init (void) {
}

void TrajectoryForTutorial::compute (void) {
    computeProgression () ;
    computeCurrentPosition (getPreviousTarget (),
                            getCurrentTarget (),
                            _positionBetweenTargets) ;
    computeOutputs () ;
}
void TrajectoryForTutorial::computeProgression (void) {
    if (_positionBetweenTargets >= 1) {
        _currentTarget = (_currentTarget + 1) % _numberOfTargets ;
        _positionBetweenTargets = _step ;
    } else {
        _positionBetweenTargets += _step ;
    }
    _previousTarget = ((_currentTarget + _numberOfTargets - 1) %
                      _numberOfTargets) ;
}
void TrajectoryForTutorial::computeCurrentPosition (const PsTranslation & previous, 
          const PsTranslation & current, 
          const float progression) {

    for (int i = 0 ; i < 3 ; i ++) {
        _position [i] = A2B (previous [i], current [i], progression) ;
    }
}

void TrajectoryForTutorial::computeOutputs (void) {
    positionOutput.set (_position) ;
}
PsTranslation TrajectoryForTutorial::getCurrentTarget (void) {
    return (_targetsCoordinates [_currentTarget]) ;
}

PsTranslation TrajectoryForTutorial::getPreviousTarget (void) {
    return (_targetsCoordinates [_previousTarget]) ;
}

float TrajectoryForTutorial::A2B (const float & a,
                        const float & b,
                        const float & f) const {
    return (a + (b - a) * f) ;
}
#ifndef TrackerForTutorialHEADER
#define TrackerForTutorialHEADER

#include <PsSimulatedObject.h>
#include <PsvMechanismPartner.h>
#include <PsInput.h>
#include <PsTranslation.h>
#include <PsHPRRotation.h>
#include <PsPolatorAndPsNumericType.h>

class TrackerForTutorial : public PsSimulatedObject,
                           public PsvMechanismPartner  {

public :

TrackerForTutorial (PsController & ctrl,
    const PsObjectDescriptor & objectDescriptor) ;
virtual ~TrackerForTutorial (void) ;

virtual void init (void) ;
virtual void compute (void) ;

protected :

virtual void computeInputs (void) ;
virtual void computeParameters (void) ;
virtual void computeOutputs (void) ;
PsInput<PsTranslation> & targetInput;
PsControlParameter<PsTranslation> & targetParameter;
PsControlParameter<PsTranslation> & positionParameter;
PsControlParameter<PsHPRRotation> & orientationParameter;
PsOutput<PsTranslation> & positionOutput;
PsOutput<PsHPRRotation> & orientationOutput;
PsTranslation direction;
PsControlParameter<PsDouble> & slowingFactorParameter;

};

#endif
#include <TrackerForTutorial.h>
#include <PsConfigurationParameterDescriptor.h>
#include <math.h>
#include <PsPosition.h>
#include <PsvTranslationInputHandler.h>
#include <PsvHPRRotationInputHandler.h>
TrackerForTutorial::TrackerForTutorial (PsController & ctrl, const PsObjectDescriptor & objectDescriptor)
: PsSimulatedObject (ctrl, objectDescriptor), PsvMechanismPartner (),
targetInput (addInput<PsTranslation> ("target")),
positionOutput (addOutput<PsTranslation> ("position",
    new PsPolator<PsTranslation> ()))",
orientationOutput (addOutput<PsHPRRotation> ("orientation",
    new PsPolator<PsHPRRotation> ())),
targetParameter (addControlParameter<PsTranslation> ("target")),
positionParameter (addControlParameter<PsTranslation> ("position")),
orientationParameter (addControlParameter<PsHPRRotation> ("orientation")) {
visualiseOutput<PsVTranslationInputHandler> (positionOutput, "DCS_position");
visualiseOutput<PsVHPRRotationInputHandler> (orientationOutput,
    "DCS_orientation");

positionParameter.set (PsTranslation ());
orientationParameter.set (PsHPRRotation ());
slowingFactorParameter.set (9);
computeOutputs ();
}

TrackerForTutorial::~TrackerForTutorial (void) {
}
void TrackerForTutorial::init (void) {
  bool connected (false) ;
  if (getConfigurationParameters () != NULL) {
    const PsConfigurationParameterDescriptor * configParamDescriptor =
        getConfigurationParameters () -> getSubDescriptorByName ("target") ;
    // syntax : target [ targetName targetOutputName ]
    if (configParamDescriptor != NULL) {
      if (configParamDescriptor -> getNumberOfSubItems () == 2) {
        connected = targetInput . connect ( PsName ( configParamDescriptor ->
          getSubDescriptorByPosition (0) -> getAssociatedString ()),
          PsName ( configParamDescriptor ->
          getSubDescriptorByPosition (1) -> getAssociatedString ()) ) ;
      }
    }
  }
}
if (getConfigurationParameters ()->getSubDescriptorByName ("direction")
   != NULL) {
    for (int i = 0 ; i < 3 ; i ++) {
        PsFloat temp;
        std::istringstream is (getConfigurationParameters ()
            ->getSubDescriptorByName ("direction")
            ->getSubDescriptorByPosition (i)->getAssociatedString ().c_str ()) ;
        is >> temp ;
        direction [i] = temp ;
    }
} else {
    direction = PsTranslation (1, 1, 1) ;
}
if (getConfigurationParameters ()
    ->getSubDescriptorByName ("slowingFactor") != NULL) {
    slowingFactorParameter.set (atof (getConfigurationParameters ()
        ->getSubDescriptorByName ("slowingFactor")
        ->getAssociatedString ().c_str ()));
}

if (! connected) {
    cerr << "TrackerForTutorial::init (void) " << getName() << " must possess a valid target configuration parameter" << endl ;
    assert (connected);
}
void TrackerForTutorial::compute (void) {
    computeInputs () ;
    computeParameters () ;
    computeOutputs () ;
}

void TrackerForTutorial::computeInputs (void) {
    targetParameter.set (targetInput.get () ) ;
}

void TrackerForTutorial::computeOutputs (void) {
    positionOutput.set (positionParameter.get () ) ;
    orientationOutput.set (orientationParameter.get () ) ;
}
void TrackerForTutorial::computeParameters (void) {
    PsTranslation targetPosition ; // position of the target
    PsTranslation position ;       // position of the tracker
    PsHPRRotation orientation ;   // orientation of the tracker
    targetPosition = targetParameter.get () ;
    position = positionParameter.get () ;
    orientation = orientationParameter.get () ;
    position = ... ; // compute new position
    orientation = ... ; // compute new orientation
    positionParameter.set (position) ;
    orientationParameter.set (orientation) ;
}
Sending and receiving valued events

- Sending a valued event:

```cpp
AValue avSent (...) ;
sendValuedEvent ("destination", "valuedEventId", avSent) ;
```

- Receiving a valued event:

```cpp
bool AReceiver::processEvent (PsEvent * event) {
    if (event->eventId == "valuedEventId") {
        PsValuedEvent<AValue> * valuedEvent =
            dynamic_cast<PsValuedEvent<AValue>*>(event) ;
        if (valuedEvent != NULL) {
            AValue avReceived = valuedEvent->value ;
            ...
        }
        processed = true ;
    }
}
```
Receiving valued events within callbacks

➢ Registering the callback:

```cpp
new PsEventListenerCallback<AReceiver>(*this, &AReceiver::theValuedEventCallback, "valuedEventId") ;
```

➢ Receiving a valued event:

```cpp
bool AReceiver::theValuedEventCallback(PsEvent * event) {
    bool processed = false ;
    if (event->eventId == "valuedEventId") {
        PsValuedEvent<AValuedEvent> * valuedEvent =
            dynamic_cast<PsValuedEvent<AValuedEvent>*>(event) ;
        ... = valuedEvent->value.get... () ;
        processed = true ;
    }
    return processed ;
}
```
Types to be used within a valued event

➢ All the basic types:
✓ PsInt, PsLongInt, PsFloat, PsDouble
✓ PsString, PsName

➢ All the contrib/userTypes/event types:
✓ Ps3DVisEvent, PsPickInfo, PsXEvent

➢ All the contrib/userTypes/math types:
✓ PsTranslation, PsHPRotation, ...
✓ PsTranslationHPRRotation, PsScale, ...

➢ Any new user type extending PsType...
Creating a new data type

- Extend PsType
- Provide a default constructor
- Define some inherited pure virtual methods:
  - insertInStream
  - extract
  - createPolator
- Define some methods to optimize distribution:
  - pack
  - unpack
#ifndef EphemereEventForTutorialHEADER
#define EphemereEventForTutorialHEADER

#include <PsType.h>

class EphemereEventForTutorial : public PsType {
    public:
        EphemereEventForTutorial (void) ;
        EphemereEventForTutorial (const int points) ;
        virtual ~EphemereEventForTutorial (void) ;

        virtual const int getPoints (void) const ;
        virtual void setPoints (const int points) ;

};
virtual void insertInStream (ostream & = cout) const ;
virtual void extract (istream & = cin) ;

virtual void pack (PsOutgoingSynchronisationMessage & out) const ;
virtual void unpack (PsIncomingSynchronisationMessage & in) ;

virtual PsPolatorNT * createPolator (void) ;

protected :
    int _points ;
} ;
#include <EphemereEventForTutorial.h>
#include <PsPolator.h>

EphemereEventForTutorial::EphemereEventForTutorial (void) {
    _points = 100 ;
}

EphemereEventForTutorial::EphemereEventForTutorial (const int points) {
    _points = points ;
}

EphemereEventForTutorial::~EphemereEventForTutorial (void) {
}
const int EphemereEventForTutorial::getPoints (void) const {
    return _points ;
}

void EphemereEventForTutorial::setPoints (const int points) {
    _points = points ;
}
void EphemereEventForTutorial::insertInStream (ostream & out) const {
    out << _points << " " ;
}

void EphemereEventForTutorial::extract (istream & in) {
    in >> _points ;
}

PsPolatorNT * EphemereEventForTutorial::createPolator (void) {
    return new PsPolator<EphemereEventForTutorial> () ;
}
void EphemereEventForTutorial::pack (PsOutgoingSynchronisationMessage & out)
const {
    out << _points ; // binary streams : no spaces left between the inserted fields !
}

void EphemereEventForTutorial::unpack (PsIncomingSynchronisationMessage & in) {
    in >> _points ;
}
Sending a valued event

... else if (event->eventId == "Killed") {
    EphemereEventForTutorial eeft (10);
    sendValuedEvent (event->sender, "I have been killed by you", eeft);
    getController ().destroyObject (getName (), true);
    processed = true;
}
...

137
Receiving a valued event

... else if (eventId == "I have been killed by you") {
    PsValuedEvent<EphemereEventForTutorial> * valuedEvent =
        dynamic_cast<PsValuedEvent<EphemereEventForTutorial>*>(event);
    nbObjectsKilled++;
    nbPoints += valuedEvent->value.getPoints();
    processed = true;
}

...
Creating new OpenMASK classes

➢ Most efficient solution (linking duration...) :
  ✔ Build small shared dynamic libraries (.so)
  ✔ Copy (or link) them in a directory in the execution directory (LD_LIBRARY_PATH contains ./$PVM_ARCH)

➢ Proposed run-time architecture :
  ✔ Sources : the .h and .cxx files
    ✗ MainTP : the main.cxx and its associated Makefile
    ✗ LibTP1 : your new classes and a Makefile to generate the library
    ✔ LibTP2 : ...
  ✔ Execution : the run-time and the configuration files
    ✗ The .monop, .multi, .processX configuration files
    ✗ LINUX : the run-time file, the .o and .so files
  ✔ Somewhere else (../../Data ?) : the VRML files...
The new main Makefile (1/2)

include $(OpenMASKDIR)/make.in

COMPILE = $(COMPILER) $(COMPILEFLAGS) -g -fpermissive
LINK = $(LINKER) -Wl,-noinhibit-exec -L/local/cwiid-0.6.00/libcwiid/lib -lcwiid \ -L../LibTP/LINUX -lTP

INCDIR = -I. -I../LibTP \ -I/usr/local/EVC/cwiid-0.6.00/libcwiid/include \ $(shell $(OpenMASKDIR)/bin/omk-config --cxxflags)

ifdef PVM_ROOT
  PVMLIBS = -L$(PVM_ROOT)/lib/$(PVM_ARCH) -lOpenMASKPvm -lgpvm3 -lpvm3
  INCDIR += -I$(PVM_ROOT)/include
endif

DSO = $(shell $(OpenMASKDIR)/bin/omk-config --ldflags) \ -L$(PVM_ROOT)/lib/$(PVM_ARCH)
TARGETNAME = ./$(PVM_ARCH)/tutorial

$(TARGETNAME): $(OBJ) $(PVM_ARCH)/main.o
  $(LINK) $(OBJ) $(PVM_ARCH)/main.o $(DSO) -o $@

$(PVM_ARCH)/main.o : main.cxx
  $(COMPILE) $(INCDIR) -g -c main.cxx -o $@

all:
  make $(TARGETNAME)

clean:
  rm -rf ./$(PVM_ARCH)/*.o

cleanall:
  rm -rf ./$(PVM_ARCH)/*.o $(TARGETNAME)
include $(OpenMASKDIR)/make.in

COMPILE = $(COMPILER) $(COMPILEFLAGS) -g -fpermissive
LINK = $(LINKER) -WI,-noinhibit-exec -L/local/cwiid-0.6.00/libcwiid/lib -lcwiid

INCDIR = -I. \ 
        -I /usr/local/EVC/cwiid-0.6.00/libcwiid/include \ 
$(shell $(OpenMASKDIR)/bin/omk-config --cxxflags)

ifdef PVM_ROOT
    PVMLIBS = -L$(PVM_ROOT)/lib/$(PVM_ARCH) -lOpenMASKPvm -lgpvm3 -lpvm3
    INCDIR += -I$(PVM_ROOT)/include
endif

DSO = $(shell $(OpenMASKDIR)/bin/omk-config --ldflags) \ 
      -L$(PVM_ROOT)/lib/$(PVM_ARCH)
OBJ = ./(PVM_ARCH)/MyMovableObject.o \ 
    ./$(PVM_ARCH)/MyTargetObject.o

TARGETNAME = ./$(PVM_ARCH)/libTP.so

$(TARGETNAME): $(OBJ)
    g++ -shared $(OBJ) -o $@
    cp $(TARGETNAME) ../../Execution/$(PVM_ARCH)

$(PVM_ARCH)/%_.o : %.cxx
    $(COMPILE) $(INCDIR) -c $< -o $@

all: ...
clean: ...
cleanall: ...
Solution for data-flow distribution

➢ Entities can be distributed upon several processes
➢ These real entities are called “referentials”
➢ A mirror (a.k.a. “proxy”, “ghost”) of a real entity is created on each process where is found an object subscriber to an output of its referential
➢ The kernel of OpenMASK ensures the correct propagation of the outputs' values toward all the mirrors of an entity
Distribution – Communication

T1: Trajectory
- TrajectoryBehavior
  - Position
  - Orientation

T2: Tracker
- TrackerBehavior
  - Position
  - Orientation

M_T1: Mirror
- MirrorBehavior
  - Position

TargetPosition

Process A

Process B
Distribution – Synchronization

➢ A local controller is created on each node

➢ Algorithm parameterized by latency:
  ✔ Proceed to simulation of date $T$ only if all the update data of simulation step of date $T - dT$ - latency from the other controllers is available
  ✔ Send update values to the other local controllers
  ✔ Fast controllers have to wait for slower ones

➢ Simulation step $T$ takes place while update values of $T - dT$ transit through the network:
  ✔ Controllers do not have to wait for update values
Distribution – Consistency

➢ Extrapolation avoids waiting for exact values
  ✔ Good approximations are available for date T
  ✔ And for date T - dT if needed depending on the latency

➢ If latency is made to correspond to network latency, distribution does not slow the simulation
  ✔ Very little lag between the different machines
  ✔ But a constant delay (due to the latency) is introduced in the propagation of signal changes
Distribution for message sending

- When a message is sent to an object:
  - ✔ If its referential is in the same process:
    - ✗ OK, the message can be processed
  - ✔ If not:
    - ✗ the message is first processed by a mirror
    - ✗ this mirror can be created dynamically if needed
    - ✗ the mirror transmits the message towards the process of its associated referential
    - ✗ the reception of the message will be delayed ...
    - ✗ the messages travel across the network with the update values of the mirrors' outputs
From a programmer point of view

➢ General knowledge of VR-OpenMASK :
  ✓ The OpenMASK global architecture
  ✓ The API of a simulated object
  ✓ The functioning principles of the 3D Visualization
  ✓ The functioning principles of the distribution

➢ Develop new VR-OpenMASK entities with :
  ✓ Their own sets of inputs, control parameters, outputs
  ✓ Their own behavior
  ✓ Their own visualizable fields
From a contributor point of view

➢ How does the VR-OpenMASK interaction protocol work?
➢ What is the structure of an interactive object?
➢ What is the structure of an interaction tool?
➢ How is it possible to develop new VR-OpenMASK interaction utilities?

✔ New adapters to make an entity interactive
✔ New behaviors for existing interaction tools
✔ New interactors to create new interaction tools
Interactions with VR-OpenMASK

➢ An entity can be controlled by another one:
  ✓ Thanks to message sending
  ✓ Thanks to data-flow connections

➢ Controlled entities are called “interactive objects”

➢ Controlling entities are called “interaction tools”

➢ These objects must share a communication protocol in order to be able to interact together:
  ✓ Through this protocol, interactive objects can be manipulated by different kinds of interaction tools
Protocol – Asking for interaction

Target

TargetBehavior

TargetPosition

Orientation

Position

InteractiveTracker

InteractiveTrackerBehavior

TargetBehavior

TargetPosition

Orientation

Position

TPosition

InteractorPosition

ITPosition

InteractorBehavior

User event

Control takeover request
Protocol – Accepting interaction

Target
  - TargetBehavior
    - TargetPosition
    - TargetOrientation
  - TargetBehavior
  - Position
  - Orientation

Interactor
  - InteractorPosition
  - InteractorBehavior
  - TPosition
  - TPosition

InteractiveTracker
  - ImposedPosition
  - InteractiveTrackerBehavior
    - Position
    - Orientation

Control takeover acceptance
Interacting with a tracker
Designing interactive objects

- **From scratch:**
  - Needs a good knowledge of interaction mechanisms...

- **Using C++ templates ProtocolTeacher classes:**
  - Knowledge of the interaction protocol
  - In relation with a ConnectorProvider that:
    - Provides a connector to use with an interactor
    - Can switch connectors
  - The Connector does the real work:
    - Simple or double parameters, constrained, multi-users...
Making interactive a tracker (1/6)
controller->addInstanceCreator ("TrackerForTutorial",
    new PsSimpleSimulatedObjectCreator<DoubleProtocolTeacher
        <TrackerForTutorial, PsTranslation, PsHPRRotation> > () );
Making interactive a tracker (3/6)

tracker1 {
    Class TrackerForTutorial
    Scheduling {
        Frequency 75
        Process visualProcessA
    }
    UserParams {
        // Tracker's parameters
        geometryFileName ../../Data/RedCone.wrl
        direction [0 1 0] // RedCone avance vers Y+
        target [trajectory position]
        slowingFactor 100
    }
}
Making interactive a tracker (4/6)

// DoubleConnector's parameters
connectorClassName SimpleConnector
controlParameterClassNames [ PsTranslation ]
controlParameterNames [ position ]
interactorOutputNames [ interactorPosition ]
enableAwareness yes
adaptorAwarenessKind [ boundingBox ]
adaptorInteractiveAreaAwareness yes
adaptorAwarenessNode DCS_position

}
Making interactive a tracker (5/6)

tracker2 {
   Class TrackerForTutorial
   Scheduling {
      Frequency 75
      Process visualProcessA
   }
   UserParams {
      geometryFileName ../../Data/TieFighter.wrl
      direction [0 0 1] // TieFighter avance vers Z+
      target [tracker1 position]
      slowingFactor 100
   }
}
Making interactive a tracker (6/6)

// adaptator's parameters
connectorClassName DoubleConnector
controlParameterClassNames [ PsTranslation PsHPRRRotation]
controlParameterNames [ position orientation ]
interactorOuputNames [ interactorPosition interactorOrientation ]
enableAwareness yes
adaptorAwarenessKind [ boundingBox ]
adaptorInteractiveAreaAwareness yes
adaptorAwarenessNode DCS_position

}
Designing a new type to be interactive

➢ Must follow some rules:

✔ The compute method must only call three protected methods:
  ✗ computeInputs
  ✗ computeParameters
  ✗ computeOutputs

✔ The datas that will be used for interaction must be publicly published:
  ✗ they must be control parameters
#ifndef MotionLessForTutorialHEADER
#define MotionLessForTutorialHEADER

#include <PsSimulatedObject.h>
#include <PsvMechanismPartner.h>
#include <PsOutput.h>
#include <PsControlParameter.h>
#include <PsTranslation.h>
#include <PsHPRRotation.h>

class MotionLessForTutorial : public PsSimulatedObject,
                           public PsvMechanismPartner {

public:

MotionLessForTutorial (PsController & ctrl,
const PsObjectDescriptor & objectDescriptor);
virtual ~MotionLessForTutorial (void);
virtual void init (void);
virtual void compute (void);

protected:

virtual void computeInputs (void);
virtual void computeParameters (void);
virtual void computeOutputs (void);
PsControlParameter<PsTranslation> & positionParameter;
PsWithControlParameter<PsWithHPPRRotation> & orientationParameter;

PsOutput<PsWithTranslation> & positionOutput;
PsWithOutput<PsWithHPPRRotation> & orientationOutput;

PsTranslation position;
PsWithHPPRRotation orientation;

};

#endif
#include <MotionLessForTutorial.h>
#include <PsConfigurationParameterDescriptor.h>
#include <PsvTranslationInputHandler.h>
#include <PsvHPRRotationInputHandler.h>
#include <PsSystemEventIdentifier.h>
#include <sstream>
MotionLessForTutorial::MotionLessForTutorial (PsController & ctrl,
const PsObjectDescriptor & objectDescriptor)
    : PsSimulatedObject (ctrl, objectDescriptor), PsvMechanismPartner (),
      positionOutput (addOutput<PsTranslation> ("position",
          new PsPolator<PsTranslation> ())),
      orientationOutput (addOutput<PsHPRRotation> ("orientation",
          new PsPolator<PsHPRRotation> ())),
      positionParameter (addControlParameter<PsTranslation> ("position")),
      orientationParameter (addControlParameter<PsHPRRotation> ("orientation")),
      position (0, 0, 0), orientation (0, 0, 0) {
visualiseOutput<PsvTranslationInputHandler> (positionOutput, "DCS_position") ;
visualiseOutput<PsvHPRRotationInputHandler> (orientationOutput,
    "DCS_orientation") ;
if (getConfigurationParameters () != NULL) {
    if (getConfigurationParameters ()
        ->getSubDescriptorByName ("position") != NULL) {
        // syntax : position [ x y z ]
        for (int i = 0 ; i < 3 ; i ++) {
            PsFloat temp;
            istringstream is (getConfigurationParameters ()
                ->getSubDescriptorByName ("position")
                ->getSubDescriptorByPosition (i)
                ->getAssociatedString ().c_str ()) ;
            is >> temp ;
            position [i] = temp ;
        }
    }
}
if (getConfigurationParameters ()
    =>$getSubDescriptorByName ("orientation") != NULL) {
    // syntax : orientation [ h p r ]
    for (int i = 0 ; i < 3 ; i ++) {
        PsFloat temp ;
        istringstream is (getConfigurationParameters ()
            =>$getSubDescriptorByName ("orientation")
            =>$getSubDescriptorByPosition (i)
            =>$getAssociatedString ().c_str ()) ;
        is >> temp;
        orientation [i] = temp ;
    }
}
} else {
    cerr << getName () << "*** The configuration file does not exist" << endl;
    sendEvent (getController (), PsSystemEventIdentifier::MaskStop);
}
positionParameter.set (position);
orientationParameter.set (orientation);
computeOutputs ()
}

MotionLessForTutorial::~MotionLessForTutorial (void) {
}

void MotionLessForTutorial::init (void) {

}

void MotionLessForTutorial::compute () {
  computeInputs () ;
  computeParameters () ;
  computeOutputs () ;
}
void MotionLessForTutorial::computeInputs (void) {
}

void MotionLessForTutorial::computeParameters (void) {
}

void MotionLessForTutorial::computeOutputs (void) {
    positionOutput.set (positionParameter.get ()) ;
    orientationOutput.set (orientationParameter.get ()) ;
}
Making interactive an object (1/6)
controller->addInstanceCreator ("TrackerForTutorial",
    new PsSimpleSimulatedObjectCreator<DoubleProtocolTeacher
        <TrackerForTutorial, PsTranslation, PsHPRRotation> > () );
Making interactive an object (3/6)

tracker1 {
  Class TrackerForTutorial
  Scheduling {
    Frequency 75
    Process visualProcessA
  }
  UserParams {
    // Tracker's parameters
    geometryFileName ../../Data/RedCone.wrl
    direction [0 1 0] // RedCone avance vers Y+
    target [trajectory position]
    slowingFactor 100
  }
}
// DoubleConnector's parameters
connectorClassName SimpleConnector
controlParameterClassNames [ PsTranslation ]
controlParameterNames [ position]
interactorOutputNames [ interactorPosition ]
enableAwareness yes
adaptorAwarenessKind [ boundingBox ]
adaptorInteractiveAreaAwareness yes
adaptorAwarenessNode DCS_position
}
Making Interactive an object (5/6)

tracker2 {
  Class TrackerForTutorial
  Scheduling {
    Frequency 75
    Process visualProcessA
  }
  UserParams {
    geometryFileName ../../Data/TieFighter.wrl
    direction [0 0 1]  // TieFighter avance vers Z+
    target [tracker1 position]
    slowingFactor 100
  }
}
// adaptator's parameters
connectorClassName DoubleConnector
controlParameterClassNames [ PsTranslation PsHPRRotation]
controlParameterNames [ position orientation ]
interactorOutputNames [ interactorPosition interactorOrientation ]
enableAwareness yes
adaptorAwarenessKind [ boundingBox ]
adaptorInteractiveAreaAwareness yes
adaptorAwarenessNode DCS_position
}
}
Virtual tools for interaction

- Virtual ray
- Virtual hand
- Virtual 3D cursor
Interactors are compound objects

➢ To offer maximum software reuse
➢ To be independent from physical devices
➢ To allow different behaviors for “same” devices:
  ✔ That look the same
  ✔ That are used the same way by the end-users
  ✔ But that drive differently the interactive objects
➢ In the future, it could be possible to switch the behaviors:
  ✔ The same way as for the interactive objects
Interactors - General behavior

3D Picking Service

PickingBehavior

Picker request

Picker result

Event Handler

EventHandlerBehavior

Event

User event

Device Driver

DeviceDriverBehavior

DeviceDriverBehavior

Event

...
VirtualRay utility (1/4)

- Needs a support to follow and a trigger
- When no physical device are available, needs a virtual support attached to the viewpoint and a virtual trigger:
  - SupportOnCameraman
  - XmouseTrigger
- Inherits from:
  - SupportedPandO
  - ShiftedPandO
... #include <VirtualRayForTutorial.h>
#include <RayBehaviorForTutorial.h>
...

... controller->addInstanceCreator ("VirtualRay",
    new PsSimpleSimulatedObjectCreator<
        DoubleInteractor<VirtualRayForTutorial<RayBehaviorForTutorial>,
            PsTranslation, PsHPRRotation> > ());

...
VirtualRay utility (3/4)

greenRayA {
   Class VirtualRay
   Scheduling {
      Frequency 75
      Process visualProcessA
   }
   UserParams {
      // PsvMechanismPartner's parameter
      geometryFileName ../../../Data/GreenRay.wrl
      // SupportedPandO's parameters //
      positionToFollow [ greenRaySupportA position ]
      orientationToFollow [ greenRaySupportA orientation ]
   }
}
// ShiftedPandOs's parameters //
positionOffset [ 0 0 -1 ]
orientationOffset [ 0 0 0 ]
tracePositionAndOrientation no
// VirtualRay's parameters //
associatedPicker visA
kindOfPicking 3DPick // valeur par défaut
pickingLength 400 // même longueur que la géométrie du rayon
targetPositionParameter proposedTargetPosition
targetOrientationParameter proposedTargetOrientation
interactorPositionOutput interactorPosition
interactorOrientationOutput interactorOrientation

}
SupportOnCameraman utility (1/4)

- Main features:
  - Connected to a PsTranslationHPRRotation
  - Provides a PsTranslationHPRRotation
  - Parametered by a PsTranslation offset
  - Templated by a kind of MotionLess
  - Can be made interactive with a protocol teacher:
    - to apply to the template parameter
...  
#include <SupportOnCameraman.h>  
...  
...  
controller->addInstanceCreator("SupportOnCameraman",  
    new PsSimpleSimulatedObjectCreator<  
        SupportOnCameraman<DoubleProtocolTeacher<  
            MotionLessPandO, PsTranslation, PsHPRRotation> >> ()) ;  
...  
...
greenRaySupportA {
    Class SupportOnCameraman
    Scheduling {
        Frequency 75
        Process visualProcessA
    }
    UserParams {
        // MotionLessPandO's parameters
        geometryFileName ../../Data/GreenSupport.wrl
        clone false
        tracePositionAndOrientation no
        // SupportOnCameraman's parameters
        positionOffset [ -3 -2 -10 0 0 0 ]
        positionOrientationToFollow [ cameramanA positionOrientation ]
SupportOnCameraman utility (4/4)

// DoubleConnector's parameters
controlParameterClassNames [ PsTranslation PsHPRRotation ]
controlParameterNames [ position orientation ]
interactorOuputNames [ interactorPosition interactorOrientation ]
enableAwareness yes
adaptorAwarenessKind [upScaledGeometry]
adaptorInteractiveAreaAwareness no
adaptorAwarenessNode DCS_position
XMouseTrigger utility (1/3)

- **Main features:**
  - ✔ Connected to an Interactive3DVis
  - ✔ Parametrized by:
    - ✗ the event to manage
    - ✗ the resulting event to send
  - ✔ When a this event occurs:
    - ✗ ask for a 2D picking to the Interactive3DVis
    - ✗ send the resulting event to the picked object
XMouseTrigger utility (2/3)

... include <XmouseTrigger.h> ... 

... 

controller->addInstanceCreator ("XmouseTrigger", 
    new PsSimpleSimulatedObjectCreator<XMouseTrigger> ()) ;

...
xMouseTriggerA {
    Class XMouseTrigger
    Scheduling {
        Frequency 75
        Process visualProcessA
    }
    UserParams {
        associatedPicker visA
        Events [
            { Mousekey "middle" SignalToSend "Trigger" }
        ]
    }
}
A new behavior for an interactor

➢ Provide a new behavior implementing the Behavior interface:

✔ FallBehavior

➢ Template the interactor by this new behavior:

controller->addInstanceCreator (VirtualRayWithFallBehavior,
   new PsSimpleSimulatedObjectCreator
   <DoubleInteractor
     <VirtualRay<FallBehavior>, PsTranslation, PsHPRRotation> > ()) ;
Using other behaviors
An interactor for mouse and keyboard (1/6)

➢ Graphic2DMouse : a generic interactor !
  ✔ Template parameter for DoubleInteractor
  ✔ Graphic2DMouse : inherits from VirtualDevice
  ✔ VirtualDevice : uses MoveBehavior
  ✔ MoveBehavior : inherits from AbstractBehavior

➢ Graphic2DMouse needs events :
  ✔ They can be provided by a GraphicEventHandlerFilter
An interactor for mouse and keyboard (2/6)

#include <DoubleInteractor.h>
#include <Graphic2DMouse.h>

...

controller->addInstanceCreator ("Graphic2DMouseInteractor",
   new PsSimpleSimulatedObjectCreator<
      DoubleInteractor<Graphic2DMouse, PsTranslation, PsHPRRotation> > ()
   )

...
An interactor for mouse and keyboard (3/6)

OBJ = ./$(PVM_ARCH)/Cameraman.o \
  ./$(PVM_ARCH)/GraphicEventHandlerFilter.o \n  ./$(PVM_ARCH)/TrajectoryForTutorial.o \n  ./$(PVM_ARCH)/TrackerForTutorial.o \n  ./$(PVM_ARCH)/VirtualDevice.o \n  ./$(PVM_ARCH)/AbstractMoveBehavior.o \n  ./$(PVM_ARCH)/MoveBehavior.o \n  ./$(PVM_ARCH)/Graphic2DMouse.o \n $(NULL)
An interactor for mouse and keyboard (4/6)

mouseGraphicEventHandlerFilterA {
  Class GraphicEventHandlerFilter
  Scheduling {
    Frequency 75
    Process visualProcessA
  }
  UserParams {
    VisualisationName visA
    PressEvents [ 
      { Key "Shift_L" Signal "Trigger" To mouseA }
      { Key "Ctrl_L" Signal "ComputeInLocalCoordinates" To mouseA }
      { Key "Alt_L" Signal "ComputeInGlobalCoordinates" To mouseA }
      { Key "Button1" Signal "ActivateTranslationComputation" To mouseA }
      { Key "Button2" Signal "ActivateRotationComputation" To mouseA }
      { Key "Button3" Signal "ActivateOrthogonalCompute" To mouseA } ]
}
ReleaseEvents [  
  { Key "Ctrl_L" Signal "DoNotComputeInLocalCoordinates" To mouseA }  
  { Key "Alt_L" Signal "DoNotComputeInGlobalCoordinates" To mouseA }  
  { Key "Button1" Signal "DesactivateTranslationComputation" To mouseA }  
  { Key "Button2" Signal "DesactivateRotationComputation" To mouseA }  
  { Key "Button3" Signal "DesactivateOrthogonalCompute" To mouseA }  
]
mouseA {
    Class Graphic2DMouseInteractor
    Scheduling {
        Frequency 75
        Process visualProcessA
    }
    UserParams {
        associatedPicker visA
        referential [ cameramanA positionOrientation ]
        targetPositionParameter   proposedTargetPosition
        targetOrientationParameter   proposedTargetOrientation
        interactorPositionOutput   interactorPosition
        interactorOrientationOutput   interactorOrientation
    }
}
An adapter for multi-user interactions

➢ We have to create 3 new classes:
  ✓ DoubleMultipleProtocolTeacher
  ✓ DoubleMultipleConnectorProvider
  ✓ DoubleMultipleConnector
An adapter for multi-user interactions

- **DoubleMultipleProtocolTeacher**:  
  - ✔ Inherits from the DoubleProtocolTeacher  
  - ✔ Overrides the processing of the ControlTakeover message:  
    - ✗ to allow several interactors at the same time  
  - ✔ Overrides the type of the associated connector provider:  
    - ✗ doubleMultipleConnectorProvider  
  - ✔ Overrides the type of the associated default connector:  
    - ✗ doubleMultipleConnector
An adapter for multi-user interactions

- **DoubleMultipleConnectorProvider** :
  - ✔ Inherits from DoubleConnectorProvider
  - ✔ Overrides the changeConnector method :
    - ✗ to provide a DoubleMultipleConnector

- **DoubleMultipleConnector** :
  - ✔ Inherits from DoubleConnector
  - ✔ Allows several interactors to interact at the same time :
    - ✗ combines the proposed values to override the values of two control parameters (position and orientation)
A new kind of interactor

- **DoubleMultipleInteractor**:
  - ✔ Dedicated to multi-users interactions
  - ✔ Inherits from DoubleInteractor
  - ✔ Creates 2 new outputs:
    - ✗ the absolute values proposed to the object in interaction
  - ✔ Creates a new simulated object:
    - ✗ dedicated to show the difference between the place the object in interaction is and the place where it should be if there was only one tool in interaction with it
Using the MultipleInteractor
From a contributor point of view

➢ Understand existing solutions for interaction:
  ✔ The structure of an interactive object
  ✔ The structure of an interaction tool
  ✔ Their communication protocol

➢ Propose new interaction solutions:
  ✔ New adapters to make an entity interactive
    ✗ with new interaction protocols
  ✔ New behaviors to modify existing interaction tools
  ✔ New interactors dedicated to the new protocols
To conclude...

- **VR-OpenMASK allows to share interactive CVEs:**
  
  - ✔ Without programmation:
    - ✗ using or customizing existing applications
  
  - ✔ With very little programmation:
    - ✗ creating new applications relying on existing modules
  
  - ✔ With programmation:
    - ✗ creating new modules
  
  - ✔ With advanced programmation:
    - ✗ creating new utilities